Problem 4 (5 points)

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from scipy.stats import mode
from sklearn.linear_model import LogisticRegression
```

One-vs-One Multinomial Classification

Load Dataset

(Don't edit this)

- (x,y) values are stored in rows of `xy`
- class values are in `c`

In [6]:
```
x = np.array([7.4881350392732475,16.351893663724194,22.427633760716436,29.048831
y = np.array([0.11120957227224215,0.1116933996874757,0.14437480785146242,0.11818
xy = np.vstack([x,y]).T
c = np.array([0,2,2,2,2,2,0,2,2,2,2,2,0,0,2,0,1,2,0,0,1,1,1,2,0,1,0,1,1,1,0,0,1,
```

Binomial classification function

You are given a function that performs binomial classification by using sklearn's `LogisticRegression` tool: `classify = get_binomial_classifier(xy, c, A, B)`

To use it, input:

- `xy` , an array in which each row contains (x,y) coordinates of data points
- `c` , an array that specifies the class each point in `xy` belongs to
- `A` , the class of the first group (0, 1, or 2 in this problem)
- `B` , the class of the second group (0, 1, or 2 in this problem), but different from `A`

The function outputs a classifier function (`classify()` in this case), used to classify any new `xy` into group A or B, such as by using `classify(xy)` .

In [7]:
```
def get_binomial_classifier(xy, c, A, B):
    assert A != B
    xyA, xyB = xy[c==A], xy[c==B]
    cA, cB = c[c==A], c[c==B]
    model = LogisticRegression()
    xy_new = np.concatenate([xyA, xyB], 0)
    c_new = np.concatenate([cA, cB], 0)
    model.fit(xy_new,c_new)

    def classify(xy):
        pred = model.predict(xy)
        return pred

    return classify
```

Coding a 1v1 classifier

Now you will create a one-vs-one classifier to do multinomial classification. This will generate binomial classifiers for each pair of classes in the dataset. Then to predict the class of a new point, classify it using each of the binomial classifiers, and select the majority winner as the class prediction.

Complete the two functions we have started:

- `generate_all_classifiers(xy, c)` which returns a list of binary classifier functions for all possible pairs of classes (among 0, 1, and 2 in this problem)
- `classify_majority(classifiers, xy)` which loops through a list of classifiers and gets their predictions for each point in `xy`. Then using a majority voting scheme at each point, return the overall class predictions for each point.

In [8]:
```
def generate_all_classifiers(xy, c):
    # YOUR CODE GOES HERE
    # Use get_binomial_classifier() to get binomial classifiers for each pair of
    # and return a list of these classifiers

    classifiers=[]

    classifier_1 = get_binomial_classifier(xy,c,0,1)
    classifiers.append(classifier_1)

    classifier_2 = get_binomial_classifier(xy,c,1,2)
    classifiers.append(classifier_2)

    classifier_3 = get_binomial_classifier(xy,c,0,2)
    classifiers.append(classifier_3)

    return classifiers

def classify_majority(classifiers, xy):
     #YOUR CODE GOES HERE

    overall_pred = np.zeros(xy.shape[0], dtype=int)

    pred = np.zeros((xy.shape[0], len(classifiers)), dtype=int)

    for i in range(len(classifiers)):
        pred[:, i] = classifiers[i](xy)

    for i in range(xy.shape[0]):
        preds = pred[i,:]
        majority,_ = mode(preds, keepdims = True)
        overall_pred[i] = majority

    return overall_pred
```

Trying out our multinomial classifier:

In [9]:
```
classifiers = generate_all_classifiers(xy, c)
preds = classify_majority(classifiers, xy)
accuracy = np.sum(preds == c) / len(c) * 100
print("True Classes:", c)
print(" Predictions:", preds)
print("    Accuracy:", accuracy, r"%")
```

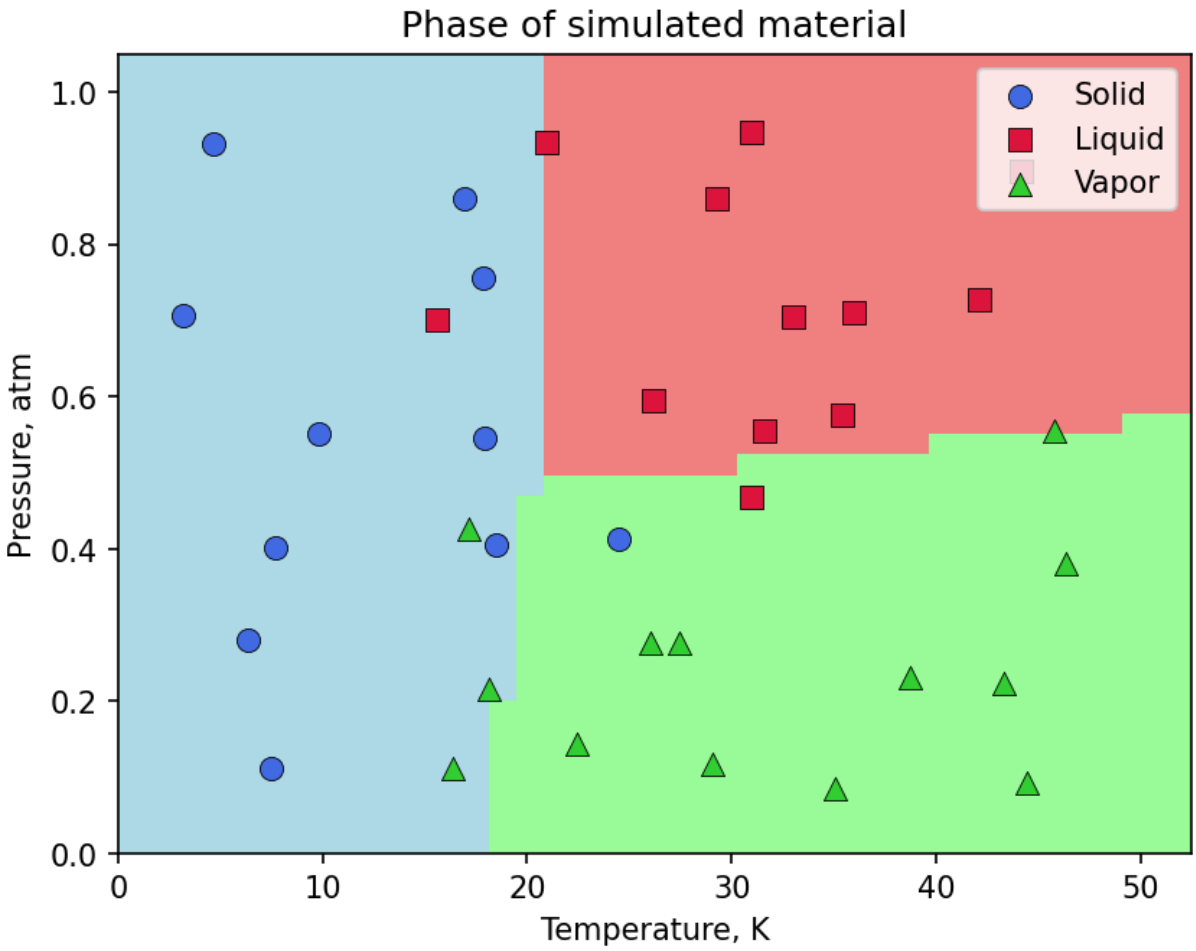
```
True Classes: [0 2 2 2 2 2 0 2 2 2 2 2 0 0 2 0 1 2 0 0 1 1 1 2 0 1 0 1 1 1 0 0
1 1 1 1]
 Predictions: [0 0 2 2 2 2 0 0 2 2 2 2 0 0 0 2 2 2 0 0 1 1 1 1 0 0 0 1 1 1 0 0
1 1 1 1]
    Accuracy: 80.55555555555556 %
```

Plotting a Decision Boundary

Here, we have made some plotting functions -- run these cells to visualize the decision boundaries.

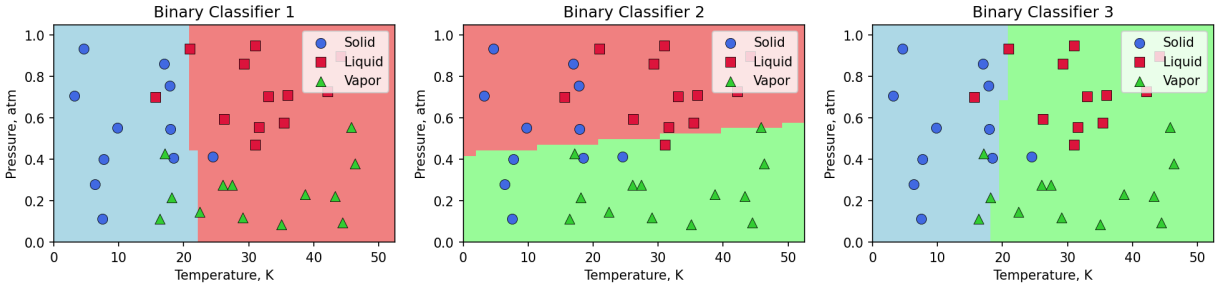
In [10]:
```
def plot_data(x, y, c, title="Phase of simulated material", newfig=True):
    xlim = [0,52.5]
    ylim = [0,1.05]
    markers = [dict(marker="o", color="royalblue"), dict(marker="s", color="crims
    labels = ["Solid", "Liquid", "Vapor"]

    if newfig:
        plt.figure(dpi=150)

    for i in range(1+max(c)):
        plt.scatter(x[c==i], y[c==i], s=60, **(markers[i]), edgecolor="black", li

    plt.title(title)
    plt.legend(loc="upper right")
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.xlabel("Temperature, K")
    plt.ylabel("Pressure, atm")
    plt.box(True)

def plot_colors(classifiers, res=40):
    xlim = [0,52.5]
    ylim = [0,1.05]
    xvals = np.linspace(*xlim,res)
    yvals = np.linspace(*ylim,res)
    x,y = np.meshgrid(xvals,yvals)
    XY = np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)
    if type(classifiers) == list:
        color = classify_majority(classifiers,XY).reshape(res,res)
    else:
        color = classifiers(XY).reshape(res,res)
    cmap = ListedColormap(["lightblue","lightcoral","palegreen"])
    plt.pcolor(x, y, color, shading="nearest", zorder=-1, cmap=cmap,vmin=0,vmax=2
    return
```

In [11]:
```
plot_data(x,y,c)
plot_colors(classifiers)
plt.show()
```



We can also look at the results of each binary classifier:

In [148]:
```
plt.figure(figsize=(16,3),dpi=150)
for i in range(3):
    plt.subplot(1,3,i+1)
    plot_data(x, y, c, title=f"Binary Classifier {i+1}", newfig=False)
    plot_colors(classifiers[i])
plt.show()
```



In [ ]: